Support Vector Machine (SVM)

Yueming Wang

2016. 11. 29

Support vector machines: history

- SVMs introduced in COLT-92 by Boser, Guyon & Vapnik. Became rather popular since.
- Theoretically well motivated algorithm: developed from Statistical Learning Theory (Vapnik & Chervonenkis) since the 60s.
- Empirically good performance: successful applications in many fields (bioinformatics, text, image recognition, . . .)

Support vector machines: history II

- Several textbooks, e.g. "An introduction to Support Vector Machines" by Cristianini and Shawe-Taylor is one.
- A large and diverse community work on them: from machine learning, optimization, statistics, neural networks, functional analysis, etc.

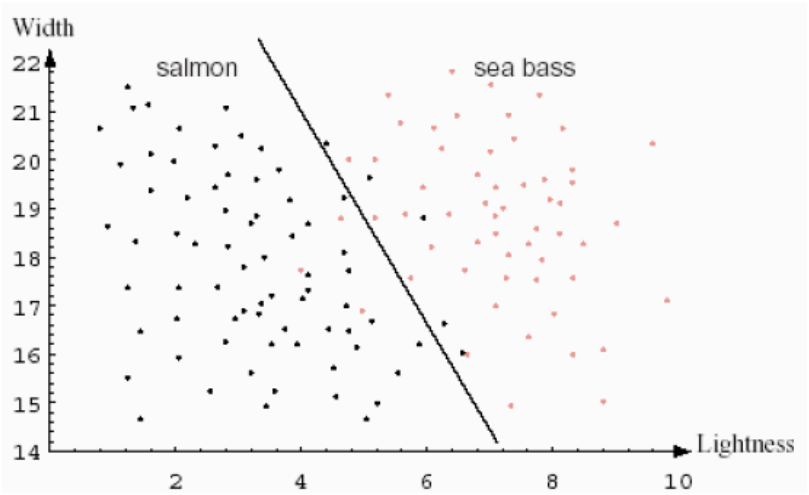
Recall pattern recognition problems

Let **x** be the input vector (observation) and y be its label:

Often, we are given a set of training data

$$S_{training} = \{(\mathbf{x}_i, y_i), i = 1..m\}$$

We use the training set to train a classifier f(**x**).



Given a set of testing data, we make the prediction of each input and evaluate the algorithm.

$$S_{testing} = \{(\mathbf{x}_i, y_i), i = 1..n\}$$

More clearly …

- input/output sets $\mathcal{X}, \mathcal{Y}$
- training set $(x_1, y_1), \ldots, (x_m, y_m)$
- "generalization": given a previously seen $x \in \mathcal{X}$, find a suitable $y \in \mathcal{Y}$.
- i.e., want to learn a classifier: $y = f(x, \alpha)$, where α are the parameters of the function.
- For example, if we are choosing our model from the set of hyperplanes in R^n, then we have:

$$f(x, \{w, b\}) = \text{sign}(w \cdot x + b).$$

Empirical risk and the true risk

- We can try to learn $f(x, \alpha)$ by choosing a function that performs well on training data:

$$R_{emp}(\alpha) = \frac{1}{m} \sum_{i=1}^{m} \ell(f(x_i, \alpha), y_i) = \text{Training Error}$$

where ℓ is the zero-one *loss function*, $\ell(y, \hat{y}) = 1$, if $y \neq \hat{y}$, and 0 otherwise. R_{emp} is called the *empirical risk*.

- By doing this we are trying to minimize the overall risk:

$$R(\alpha) = \int \ell(f(x, \alpha), y) dP(x, y) = \text{Test Error}$$

where P(x,y) is the (unknown) joint distribution function of x and y.

$$R_{emp}(\alpha) = \frac{1}{2l} \sum_{i=1}^{l} |y_i - f(\mathbf{x}_i, \alpha)|.$$

$$R(\alpha) = \int \frac{1}{2} |y - f(\mathbf{x}, \alpha)| dP(\mathbf{x}, y)$$

Choosing the set of functions

What about $f(x, \alpha)$ allowing *all* functions from $\mathcal{X}$ to $\{\pm 1\}$?

Training set $(x_1, y_1), \ldots, (x_m, y_m) \in \mathcal{X} \times \{\pm 1\}$

Test set $\bar{x_1}, \ldots, \bar{x}_{\bar{m}} \in \mathcal{X}$,

such that the two sets do not intersect.

For any f there exists f^*:

1. $f^*(x_i) = f(x_i)$ for all i
2. $f^*(x_j) \neq f(x_j)$ for all j

Based on the training data alone, there is no means of choosing which function is better. On the test set however they give different results. So generalization is not guaranteed.

$\Longrightarrow$ a restriction must be placed on the functions that we allow.

Empirical risk and the true risk

Vapnik & Chervonenkis showed that an upper bound on the true risk can be given by the empirical risk + an additional term: with probability $1 - \eta$

$$R(\alpha) \leq R_{emp}(\alpha) + \sqrt{\frac{h(log(\frac{2m}{h} + 1) - log(\frac{\eta}{4})}{m}}$$

where h is the VC dimension of the set of functions parameterized by α.

- The VC dimension of a set of functions is a measure of their *capacity* or complexity.
- If you can describe a lot of different phenomena with a set of functions then the value of h is large.

[VC dim = the maximum number of points that can be separated in all possible ways by that set of functions.]

How to compute h?

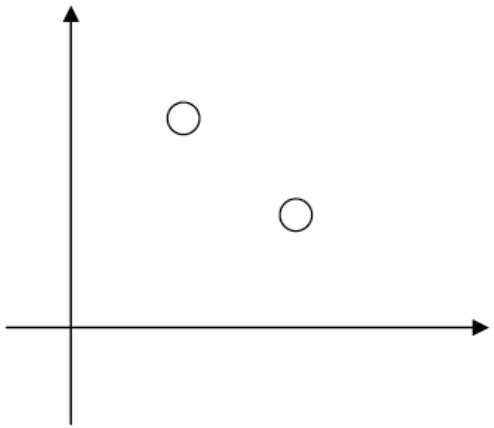
Shattering

- Machine f can shatter a set of points $\mathbf{x}_1, \mathbf{x}_2 \ldots \mathbf{x}_r$ if and only if …

  For every possible training set of the form $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_r, y_r)$, there exists some value of α that gets zero training error.

There are 2^r such training sets to consider, each with a different combinations of +1's and -1's for the y's.

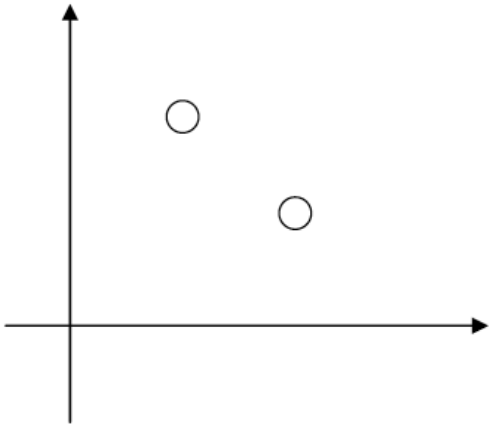
$$f(x, w) = sign(x.w)$$



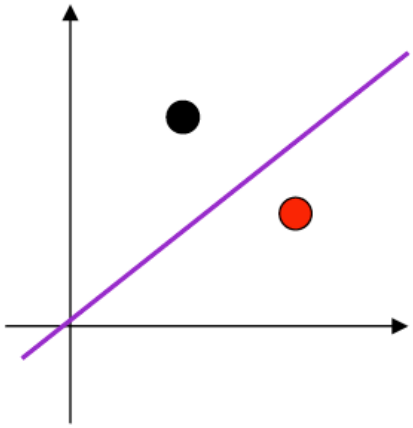
Question: Can the f(x,w) shatter the two points?

Shattering

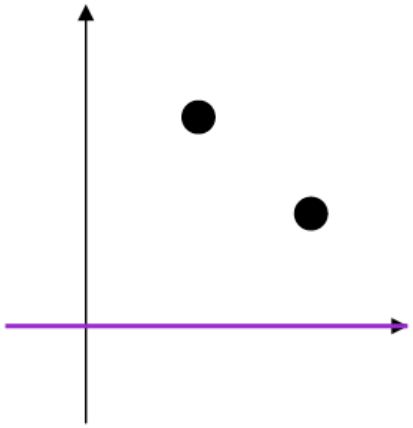
Answer: Yes.



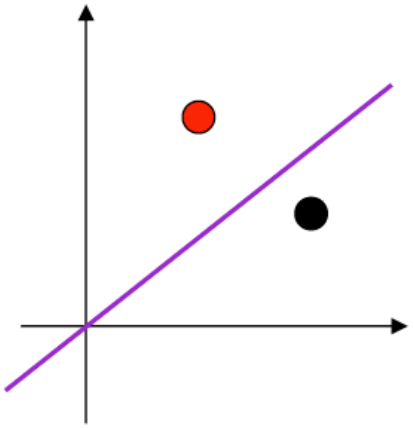
$f(x, w) = sign(x.w)$



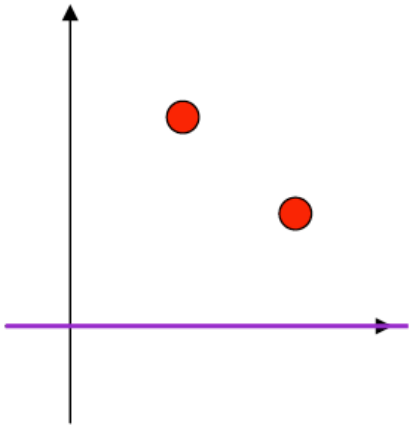
$w = (1, -1)$



$w = (0, -1)$



$w = (-1, 1)$

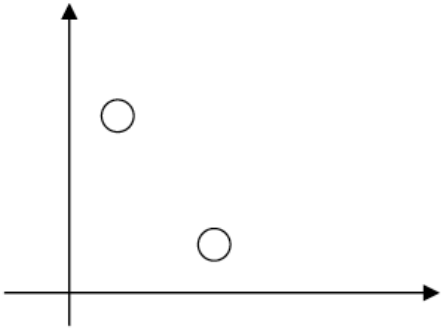


$w = (0, 1)$

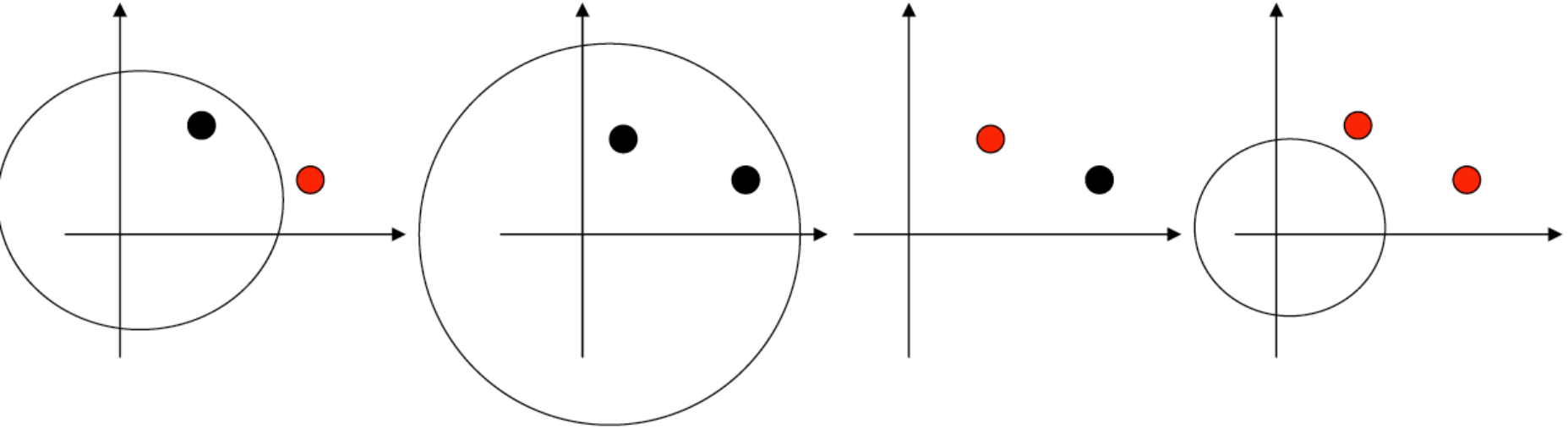
Shattering

$$f(x, b) = sign(x.x - b)$$

Question: Can the f(x,b) shatter the two points?



Answer: No.



yes yes no yes

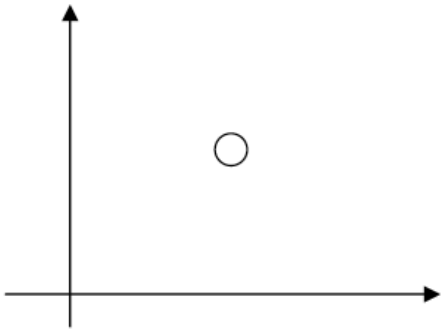
Definition of VC dimension

Given machine **f**, the VC-dimension h is:

The maximum number of points that can be arranged so that **f** shatter them.

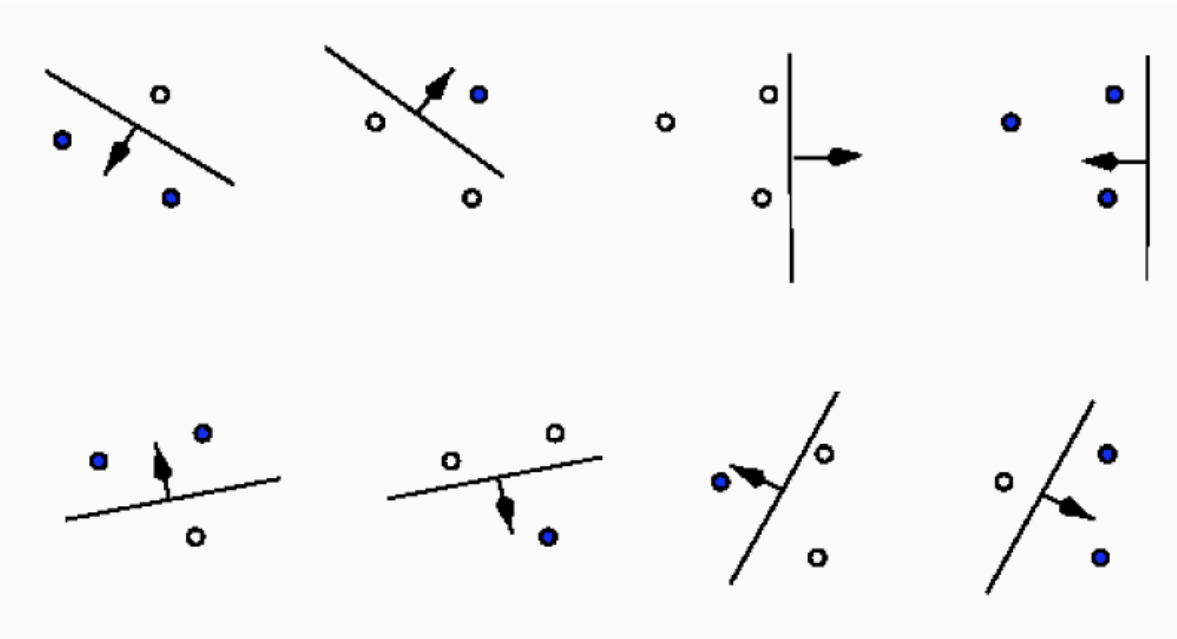
Example: what is the VC-dimension for $f(x, b) = sign(x.x - b)$

Answer: 1

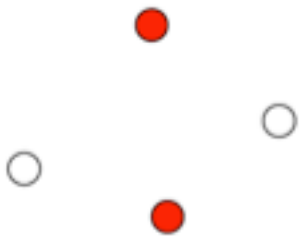


VC dimension of linear machine

$$f(x, w, b) = sign(x.w + b)$$



It's VC-dimension is 3.



VC dimension

Theory: The VC dimension of the set of oriented hyperplanes in R^n is n+1, since we can always choose n+1 points, and then choose one of the points as origin, such that the position vectors of the remaining n points are linearly independent, but can never choose n+2 such points.

What does VC-dim measure?

Is it the number of parameters?

Related but not really the same.

Proof: see Appendix of tutorial

What does VC dimension measure?

- The VC dimension of a set of functions is a measure of their *capacity* or complexity.
- If you can describe a lot of different phenomena with a set of functions then the value of h is large.

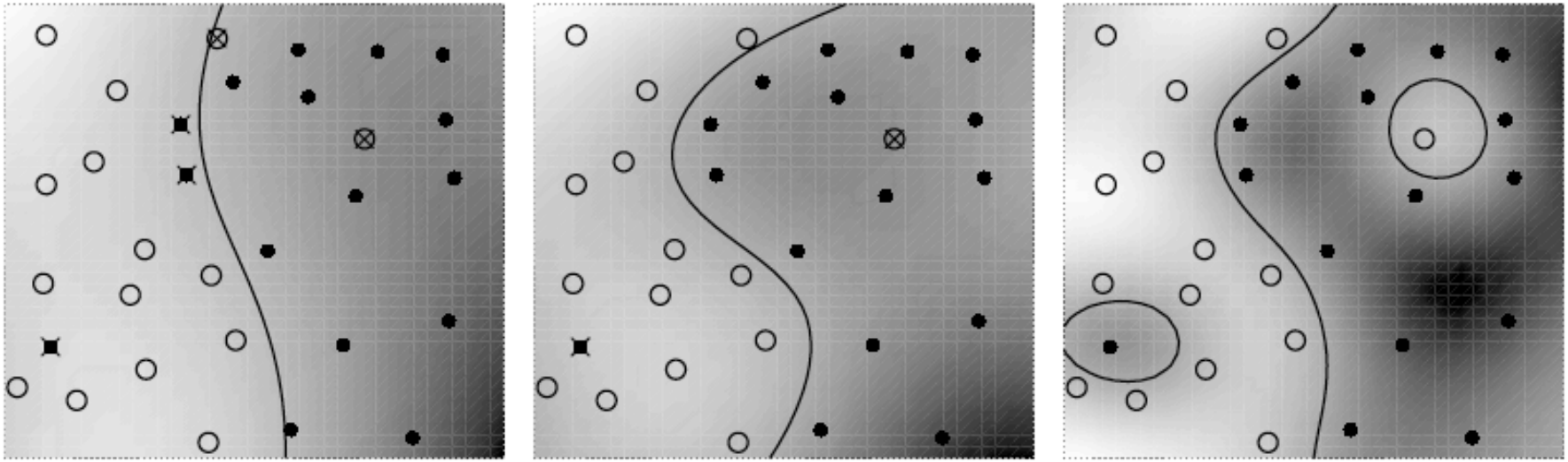
VC dimension and capacity of functions

Simplification of bound:

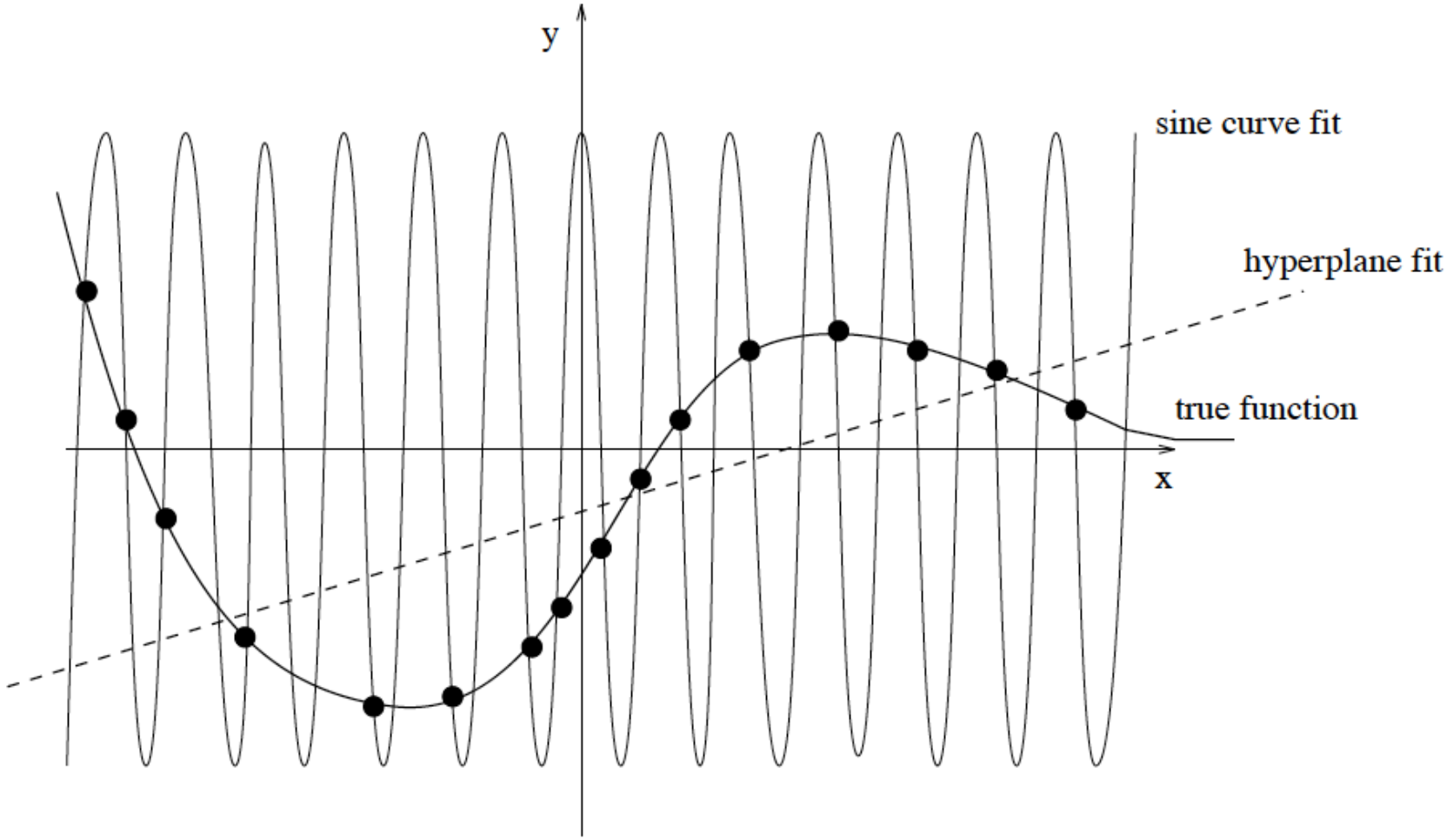
Test Error $\leq$ Training Error + Complexity of set of Models

- Actually, a lot of bounds of this form have been proved (different measures of capacity). The complexity function is often called a regularizer.
- If you take a high capacity set of functions (explain a lot) you get low training error. But you might ”overfit”.
- If you take a very simple set of models, you have low complexity, but won’t get low training error.

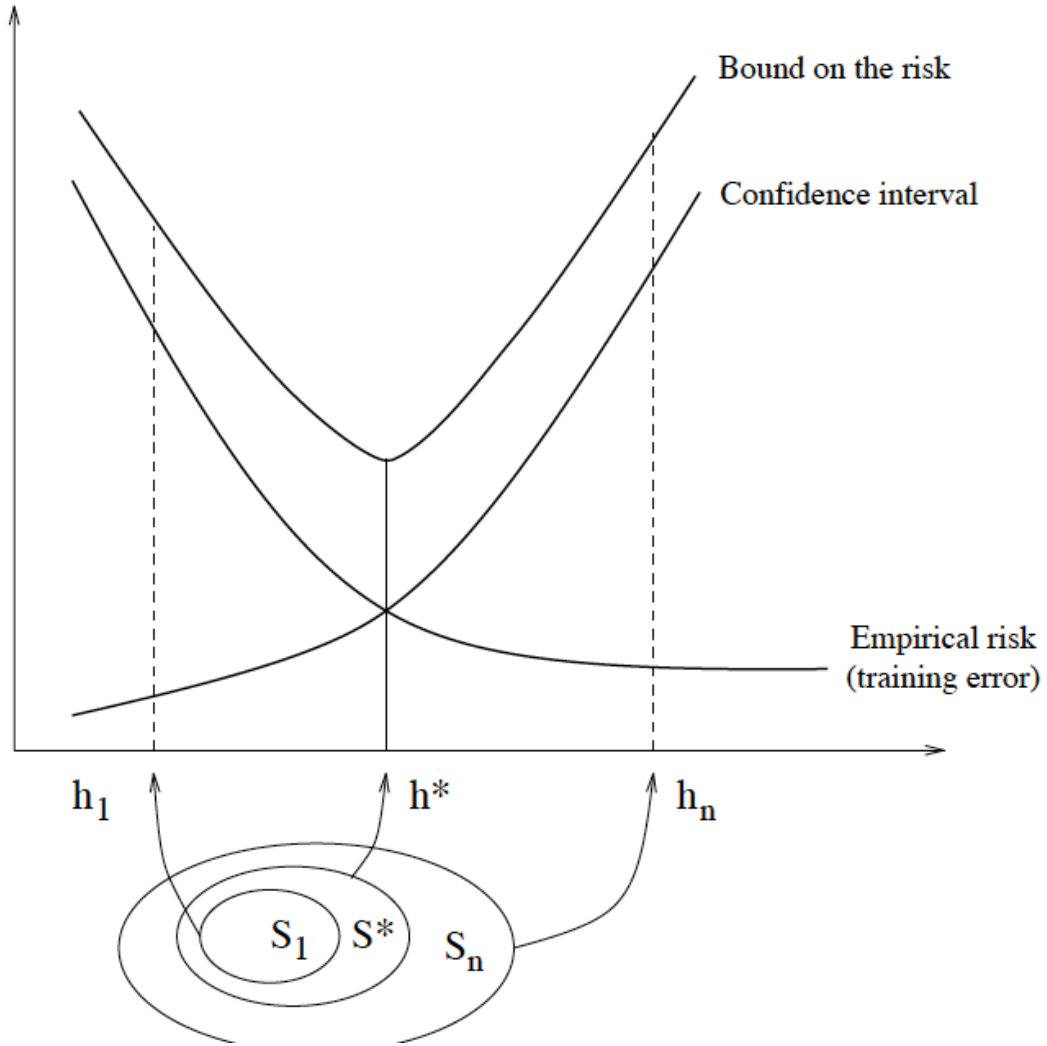
Capacity of a set of functions (classification)



Capacity of a set of functions (regression)



Structural risk minimization



Linear Support Vector Machines

So, we would like to find the function which minimizes an objective like:

Training Error + Complexity term

We write that as:

$$\frac{1}{m}\sum_{i=1}^{m}\ell(f(x_i,\alpha),y_i) + \text{Complexity term}$$

For now we will choose the set of hyperplanes (we will extend this later), so $f(x,\{w,b\}) = \text{sign}(w\cdot x + b)$

$$\frac{1}{m}\sum_{i=1}^{m}\ell(f(x,\{w,b\}),y_i) + ||w||^2$$

subject to $\min_i |w\cdot x_i + b| = 1$

Linear Support Vector Machines - separable

That function before was a little difficult to minimize because of the step function in $\ell(y, \hat{y})$ (either 1 or 0).

Let's assume we can separate the data perfectly. Then we can optimize the following:

Minimize $||w||^2$, subject to:

$$(w \cdot x_i + b) \geq 1, \text{ if } y_i = 1$$

$$(w \cdot x_i + b) \leq -1, \text{ if } y_i = -1$$

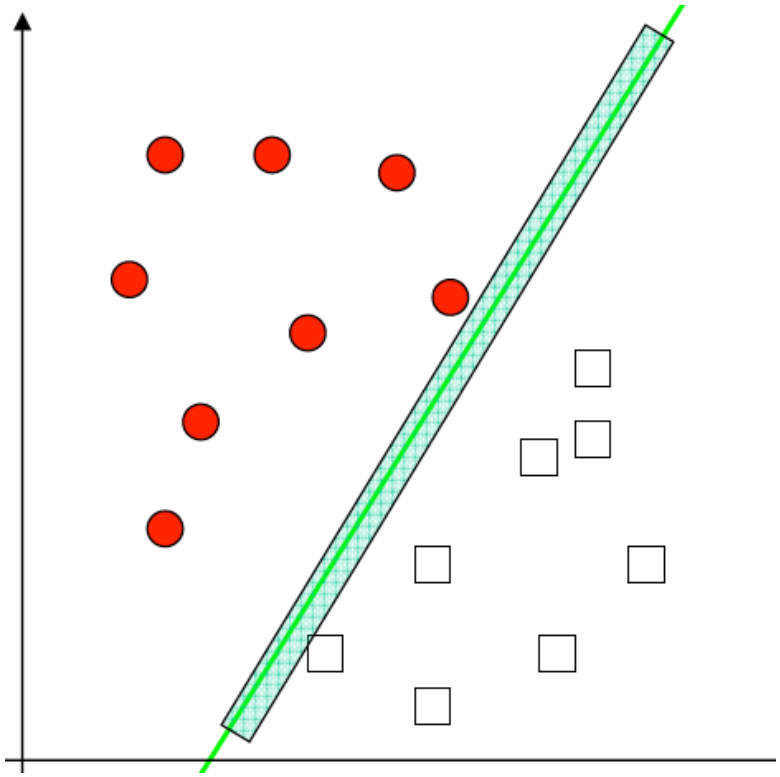
The last two constraints can be compacted to:

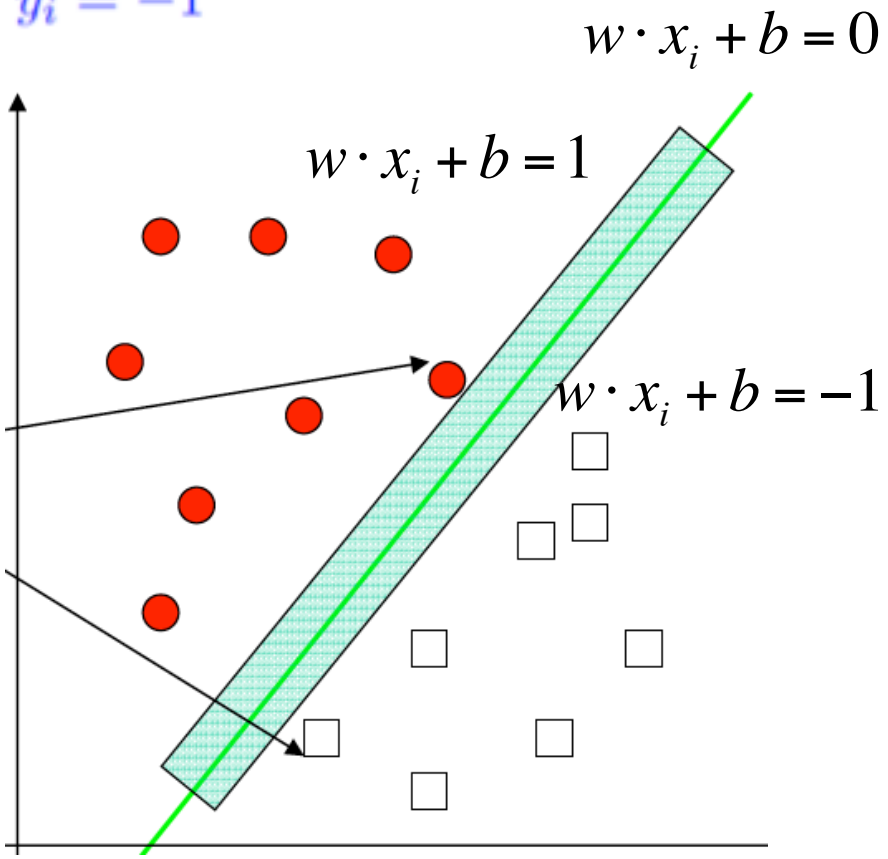
$$y_i(w \cdot x_i + b) \geq 1$$

Linear SVM – what does it mean?

$$(w \cdot x_i + b) \geq 1, \text{ if } y_i = 1$$

$$(w \cdot x_i + b) \leq -1, \text{ if } y_i = -1$$





This can always be satisfied by choosing a scale for w and b. Note that more than one w and b meets the requirement.

Linear SVM – what does it mean?

Define the margin as the width could be increased by before hitting a point. And find a w and b to maximize the margin. This feels safest. ☺

Then, how to compute the margin?

The distance from H1 to the origin is: |b+1|/|w|

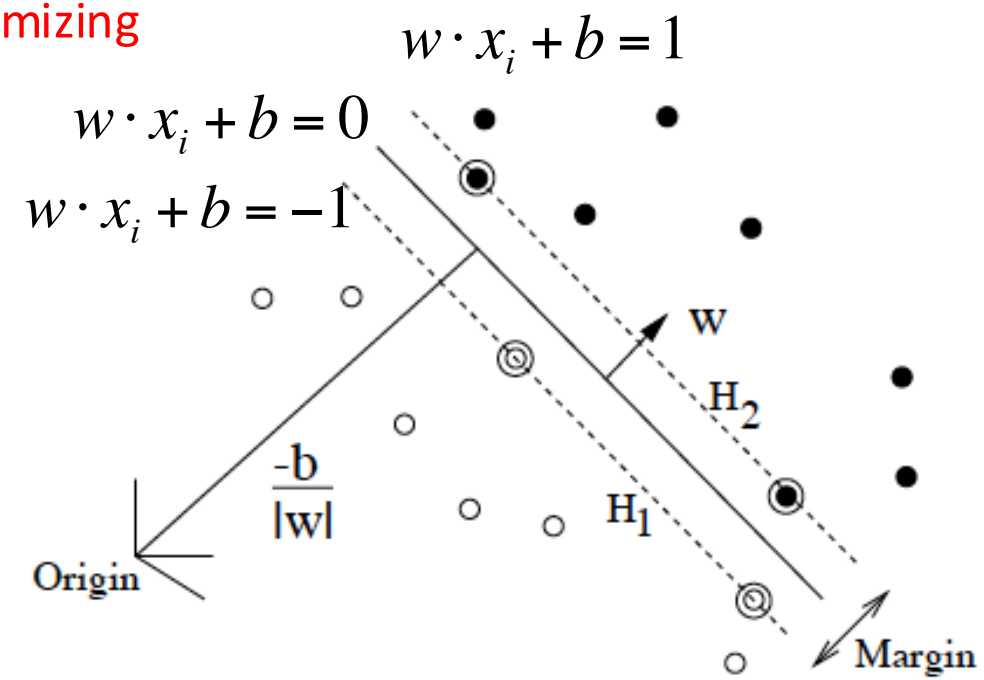
The distance from H1 to the origin is: |b-1|/|w|

The distance from the hyerplane to the origin is: |b|/|w|

So the margin is 2/|w|, maximizing margin is equivalent to:

$$\text{Minimize } ||w||^2$$

The data points used to for margin are called support vectors



Linear SVM – put everything together

$$\min_{w,b} \frac{1}{2}\|w\|^2$$

$$\text{s.t. } y_i(w^T x_i + b) \geq 1, i = 1,\ldots,m$$

The above is an optimization problem with convex quadratic objective and only linear constrains. This quadratic program can be solved using commercial quadratic programming (QP) code. However,

While we could call the problem solved here, what we will instead do is make a digression to talk about Lagrange duality. This will lead us to our optimization problem's dual form which will play a key role in allowing us to use kernels to get optimal margin classifiers to work efficiently in very high dimensional spaces. The dual form will also allow us to derive an efficient algorithm for solving the above optimization problem that will typically do much better than generic QP software.

Prerequisite: Lagrange duality

Consider a problem of the following form:

$$\begin{aligned} \min_w \quad & f(w) \\ \text{s.t.} \quad & h_i(w) = 0, \quad i = 1, \ldots, l. \end{aligned}$$

define the **Lagrangian** to be

$$\mathcal{L}(w, \beta) = f(w) + \sum_{i=1}^{l} \beta_i h_i(w)$$

Here, the β_i's are called the **Lagrange multipliers**. We would then find and set $\mathcal{L}$'s partial derivatives to zero:

$$\frac{\partial \mathcal{L}}{\partial w_i} = 0; \quad \frac{\partial \mathcal{L}}{\partial \beta_i} = 0,$$

and solve for w and β.

Prerequisite: Lagrange duality (primal problem)

Consider the following, which we'll call the **primal** optimization problem:

$$\begin{aligned} \min_w \quad & f(w) \\ \text{s.t.} \quad & g_i(w) \leq 0, \quad i = 1, \ldots, k \\ & h_i(w) = 0, \quad i = 1, \ldots, l. \end{aligned}$$

To solve it, we start by defining the **generalized Lagrangian**

$$\mathcal{L}(w, \alpha, \beta) = f(w) + \sum_{i=1}^{k} \alpha_i g_i(w) + \sum_{i=1}^{l} \beta_i h_i(w).$$

Here, the α_i's and β_i's are the Lagrange multipliers. Consider the quantity

$$\theta_{\mathcal{P}}(w) = \max_{\alpha, \beta : \alpha_i \geq 0} \mathcal{L}(w, \alpha, \beta).$$

if either $g_i(w) > 0$ or $h_i(w) \neq 0$

$$\begin{aligned} \theta_{\mathcal{P}}(w) &= \max_{\alpha, \beta : \alpha_i \geq 0} f(w) + \sum_{i=1}^{k} \alpha_i g_i(w) + \sum_{i=1}^{l} \beta_i h_i(w) \qquad (1) \\ &= \infty. \qquad (2) \end{aligned}$$

Prerequisite: Lagrange duality (primal problem)

$$\mathcal{L}(w, \alpha, \beta) = f(w) + \sum_{i=1}^{k} \alpha_i g_i(w) + \sum_{i=1}^{l} \beta_i h_i(w).$$

$$\theta_{\mathcal{P}}(w) = \max_{\alpha, \beta : \alpha_i \geq 0} \mathcal{L}(w, \alpha, \beta).$$

$$\theta_{\mathcal{P}}(w) = \begin{cases} f(w) & \text{if } w \text{ satisfies primal constraints} \\ \infty & \text{otherwise.} \end{cases}$$

Hence, if we consider the minimization problem

$$\min_w \theta_{\mathcal{P}}(w) = \min_w \max_{\alpha, \beta : \alpha_i \geq 0} \mathcal{L}(w, \alpha, \beta),$$

we see that it is the same problem (i.e., and has the same solutions as) our original, primal problem. For later use, we also define the optimal value of the objective to be $p^* = \min_w \theta_{\mathcal{P}}(w)$; we call this the **value** of the primal problem.

Prerequisite: Lagrange duality (dual problem)

Now, let's look at a slightly different problem. We define

$$\theta_{\mathcal{D}}(\alpha, \beta) = \min_{w} \mathcal{L}(w, \alpha, \beta).$$

We can now pose the **dual** optimization problem:

$$\max_{\alpha,\beta\,:\,\alpha_i \geq 0} \theta_{\mathcal{D}}(\alpha, \beta) = \max_{\alpha,\beta\,:\,\alpha_i \geq 0} \min_{w} \mathcal{L}(w, \alpha, \beta).$$

This is exactly the same as our primal problem shown above, except that the order of the "max" and the "min" are now exchanged. We also define the optimal value of the dual problem's objective to be $d^* = \max_{\alpha,\beta\,:\,\alpha_i \geq 0} \theta_{\mathcal{D}}(w)$.

How are the primal and the dual problems related? It can easily be shown that

$$d^* = \max_{\alpha,\beta\,:\,\alpha_i \geq 0} \min_{w} \mathcal{L}(w, \alpha, \beta) \leq \min_{w} \max_{\alpha,\beta\,:\,\alpha_i \geq 0} \mathcal{L}(w, \alpha, \beta) = p^*.$$

Under certain conditions, we will have,

$$d^* = p^*,$$

Prerequisite: Lagrange duality (KKT)

Karush-Kuhn-Tucker (KKT) conditions,

$$\frac{\partial}{\partial w_i}\mathcal{L}(w^*, \alpha^*, \beta^*) = 0, \quad i = 1, \ldots, n \tag{3}$$

$$\frac{\partial}{\partial \beta_i}\mathcal{L}(w^*, \alpha^*, \beta^*) = 0, \quad i = 1, \ldots, l \tag{4}$$

$$\alpha_i^* g_i(w^*) = 0, \quad i = 1, \ldots, k \tag{5}$$

$$g_i(w^*) \leq 0, \quad i = 1, \ldots, k \tag{6}$$

$$\alpha^* \geq 0, \quad i = 1, \ldots, k \tag{7}$$

Moreover, if some w^*, α^*, β^* satisfy the KKT conditions, then it is also a solution to the primal and dual problems.

For why? See: Yueming Wang, Notes of Learning How to Solve SVM. 2010.

We draw attention to Equation (5), which is called the KKT **dual complementarity** condition. Specifically, it implies that if $\alpha_i^* > 0$, then $g_i(w^*) = 0$. (I.e., the "$g_i(w) \leq 0$" constraint is **active**, meaning it holds with equality rather than with inequality.) Later on, this will be key for showing that the SVM has only a small number of "support vectors"; the KKT dual comple-

Linear Support Vector Machines - separable

Go back to:

$$\min_{w,b} \frac{1}{2}\|w\|^2$$

$$\text{s.t. } y_i(w^T x_i + b) \geq 1, i = 1,\ldots,m$$

We can write the constraints as

$$g_i(w) = -y^{(i)}(w^T x^{(i)} + b) + 1 \leq 0.$$

We have one such constraint for each training example. Note that from the KKT dual complementarity condition, we will have $\alpha_i > 0$ only for the training examples that have functional margin exactly equal to one (**support vectors**

the Lagrangian

$$\mathcal{L}(w, b, \alpha) = \frac{1}{2}||w||^2 - \sum_{i=1}^{m} \alpha_i \left[y^{(i)}(w^T x^{(i)} + b) - 1\right]. \qquad (8)$$

Linear Support Vector Machines - separable

the Lagrangian

$$\mathcal{L}(w,b,\alpha) = \frac{1}{2}||w||^2 - \sum_{i=1}^{m} \alpha_i \left[y^{(i)}(w^T x^{(i)} + b) - 1 \right]. \qquad (8)$$

Dual problem: $\max_{\alpha:\alpha_i \geq 0} \min_{w,b} L(w,b,\alpha)$

Setting L w.r.t. w and b to zero, we have:

$$\nabla_w \mathcal{L}(w,b,\alpha) = w - \sum_{i=1}^{m} \alpha_i y^{(i)} x^{(i)} = 0 \qquad w = \sum_{i=1}^{m} \alpha_i y^{(i)} x^{(i)}.$$

$$\frac{\partial}{\partial b}\mathcal{L}(w,b,\alpha) = \sum_{i=1}^{m} \alpha_i y^{(i)} = 0.$$

plug those back into the Lagrangian

$$\mathcal{L}(w,b,\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j (x^{(i)})^T x^{(j)}.$$

Linear Support Vector Machines - separable

We now have:

$$\mathcal{L}(w, b, \alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j (x^{(i)})^T x^{(j)}.$$

Then we need to solve the following dual problem:

$$\begin{aligned} \max_\alpha \quad & W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j \langle x^{(i)}, x^{(j)} \rangle. \\ \text{s.t.} \quad & \alpha_i \geq 0, \quad i = 1, \ldots, m \\ & \sum_{i=1}^{m} \alpha_i y^{(i)} = 0, \end{aligned}$$

If we obtain alpha, we then find optimal w:

$$w = \sum_{i=1}^{m} \alpha_i y^{(i)} x^{(i)}.$$

Further with w, we find the best b:

$$\min_{w,b} \frac{1}{2} \|w\|^2$$

$$\text{s.t. } y_i (w^T x_i + b) \geq 1, i = 1, \ldots, m$$

$$b^* = -\frac{\max_{i:y^{(i)}=-1} w^{*T} x^{(i)} + \min_{i:y^{(i)}=1} w^{*T} x^{(i)}}{2}.$$

Why?

Linear Support Vector Machines - separable

to solve W w.r.t. alpha:

$$\begin{aligned} \max_\alpha \quad & W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j \langle x^{(i)}, x^{(j)} \rangle. \\ \text{s.t.} \quad & \alpha_i \geq 0, \quad i = 1, \ldots, m \\ & \sum_{m}^{m} \alpha_i y^{(i)} = 0, \end{aligned}$$

The SMO (sequential minimal optimization) algorithm

First see Coordinate ascent:

Loop until convergence: {

For $i = 1, \ldots, m$, {

$\alpha_i := \arg\max_{\hat{\alpha}_i} W(\alpha_1, \ldots, \alpha_{i-1}, \hat{\alpha}_i, \alpha_{i+1}, \ldots, \alpha_m)$.

}

}

Prerequisite: SMO

$$\max_\alpha \quad W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j \langle x^{(i)}, x^{(j)} \rangle.$$

$$\text{s.t.} \quad \alpha_i \geq 0, \quad i = 1, \ldots, m$$

$$\sum_{i=1}^{m} \alpha_i y^{(i)} = 0,$$

suppose we want to hold $\alpha_2, \ldots, \alpha_m$ fixed, and take a coordinate ascent step

$$\alpha_1 = -y^{(1)} \sum_{i=2}^{m} \alpha_i y^{(i)}.$$

Can we only change alpha_1 in this step? NO!

Repeat till convergence {

1. Select some pair α_i and α_j to update next (using a heuristic that tries to pick the two that will allow us to make the biggest progress towards the global maximum).
2. Reoptimize $W(\alpha)$ with respect to α_i and α_j, while holding all the other α_k's $(k \neq i, j)$ fixed.

}

Prerequisite: SMO

In each step, choose two alphas:

$$\alpha_1 y^{(1)} + \alpha_2 y^{(2)} = -\sum_{i=3}^{m} \alpha_i y^{(i)} = \zeta.$$

$$\alpha_1 = (\zeta - \alpha_2 y^{(2)}) y^{(1)}.$$

Then, $W(\alpha_1, \alpha_2, \ldots, \alpha_m) = W((\zeta - \alpha_2 y^{(2)}) y^{(1)}, \alpha_2, \ldots, \alpha_m).$

General form: $a\alpha_2^2 + b\alpha_2 + c$

setting its derivative to zero

$$\alpha_2^{new} = \begin{cases} \alpha_2^{new,unclipped} & \text{if } L \leq \alpha_2^{new,unclipped} \\ L & \text{if } \alpha_2^{new,unclipped} < L \end{cases}$$

Note that alpha_2 >=0 and lies in the line. Suppose alpha_2 >=L.

Linear SVM – separable (final)

After we obtain alpha, we finally reach the optimal w and b:

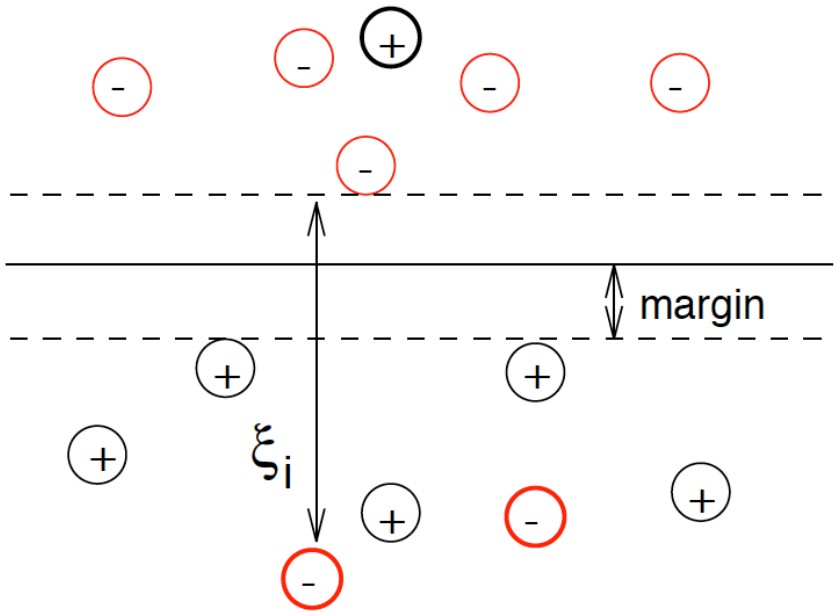
$$w = \sum_{i=1}^{m} \alpha_i y^{(i)} x^{(i)}.$$

$$b^* = -\frac{\max_{i:y^{(i)}=-1} w^{*T} x^{(i)} + \min_{i:y^{(i)}=1} w^{*T} x^{(i)}}{2}.$$

Linear Support Vector Machines – non-separable

We would like to relax the constraints (10) and (11), but only when necessary, that is, we would like to introduce a further cost (i.e. an increase in the primal objective function) for doing so. This can be done by introducing positive slack variables $\xi_i,\ i = 1, \cdots, l$ in the constraints (Cortes and Vapnik, 1995), which then become:

$$\begin{aligned} \min_{w,b} \quad & \frac{1}{2}||w||^2 + C\sum_{i=1}^{m}\xi_i \\ \text{s.t.} \quad & y^{(i)}(w^T x^{(i)} + b) \geq 1 - \xi_i, \quad i = 1, \ldots, m \\ & \xi_i \geq 0, \quad i = 1, \ldots, m. \end{aligned}$$



Linear Support Vector Machines – non-separable

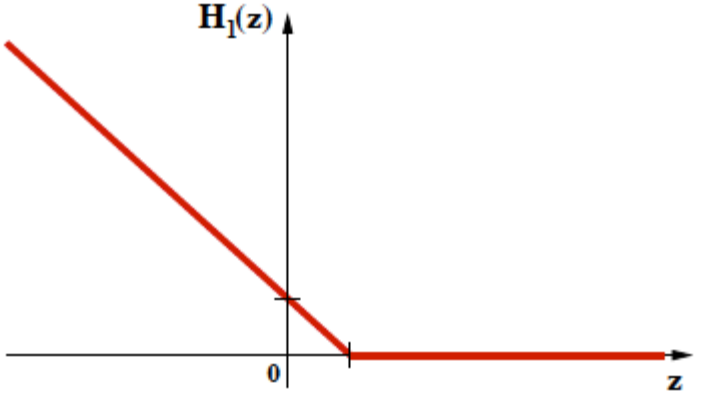
$$\min_{\gamma,w,b} \quad \frac{1}{2}||w||^2 + C\sum_{i=1}^{m}\xi_i$$
$$\text{s.t.} \quad y^{(i)}(w^T x^{(i)} + b) \geq 1 - \xi_i, \quad i = 1, \ldots, m$$
$$\xi_i \geq 0, \quad i = 1, \ldots, m.$$

Another form:

$$f(\boldsymbol{x}) = \boldsymbol{w} \cdot \boldsymbol{x} + b$$

$$\min P(\boldsymbol{w}, b) = \underbrace{\frac{1}{2}\|\boldsymbol{w}\|^2}_{\text{maximize margin}} + \underbrace{C\sum_i H_1[\, y_i\, f(\boldsymbol{x}_i)\,]}_{\text{minimize training error}}$$

Hinge Loss $H_1(z) = \max(0, 1 - z)$



Linear Support Vector Machines – non-separable

As before, we can form the Lagrangian:

$$\mathcal{L}(w, b, \xi, \alpha, r) = \frac{1}{2} w^T w + C \sum_{i=1}^{m} \xi_i - \sum_{i=1}^{m} \alpha_i \left[y^{(i)} (x^T w + b) - 1 + \xi_i \right] - \sum_{i=1}^{m} r_i \xi_i.$$

Here, the α_i's and r_i's are our Lagrange multipliers (constrained to be ≥ 0).

$$\nabla_w \mathcal{L}(w, b, \alpha) = w - \sum_{i=1}^{m} \alpha_i y^{(i)} x^{(i)} = 0 \qquad w = \sum_{i=1}^{m} \alpha_i y^{(i)} x^{(i)}.$$

$$\frac{\partial}{\partial b} \mathcal{L}(w, b, \alpha) = \sum_{i=1}^{m} \alpha_i y^{(i)} = 0. \qquad \frac{\partial L}{\partial \xi_i} = C - \alpha_i - r_i$$

So the dual form:

$$\max_{\alpha} \quad W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j \langle x^{(i)}, x^{(j)} \rangle$$

$$\text{s.t.} \quad 0 \leq \alpha_i \leq C, \quad i = 1, \ldots, m$$

$$\sum_{i=1}^{m} \alpha_i y^{(i)} = 0,$$

Linear Support Vector Machines – non-separable

$$\max_\alpha \quad W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j \langle x^{(i)}, x^{(j)} \rangle$$

$$\text{s.t.} \quad 0 \le \alpha_i \le C, \quad i = 1, \ldots, m$$

$$\sum_{i=1}^{m} \alpha_i y^{(i)} = 0,$$

the KKT dual-complementarity conditions

$$\alpha_i = 0 \quad \Rightarrow \quad y^{(i)}(w^T x^{(i)} + b) \ge 1 \tag{14}$$

$$\alpha_i = C \quad \Rightarrow \quad y^{(i)}(w^T x^{(i)} + b) \le 1 \tag{15}$$

Why?

$$0 < \alpha_i < C \quad \Rightarrow \quad y^{(i)}(w^T x^{(i)} + b) = 1. \tag{16}$$

SMO still can solve the problem. The only difference is that the range of alpha becomes [0, C]

Nonlinear Support Vector Machines – kernel tricks

Linear classifiers aren't complex enough sometimes. SVM solution:

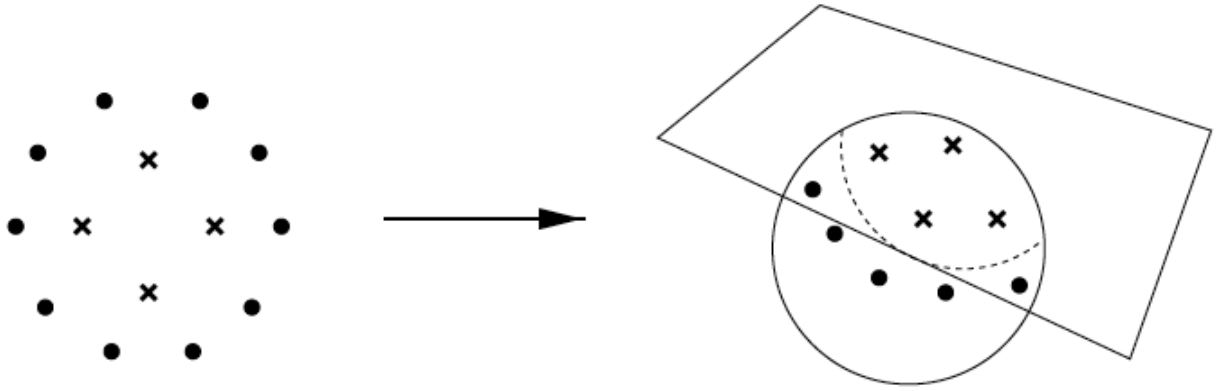
Map data into a richer feature space including nonlinear features, then construct a hyperplane in that space so all other equations are the same!

Formally, preprocess the data with:

$$x \mapsto \Phi(x)$$

and then learn the map from $\Phi(x)$ to y:

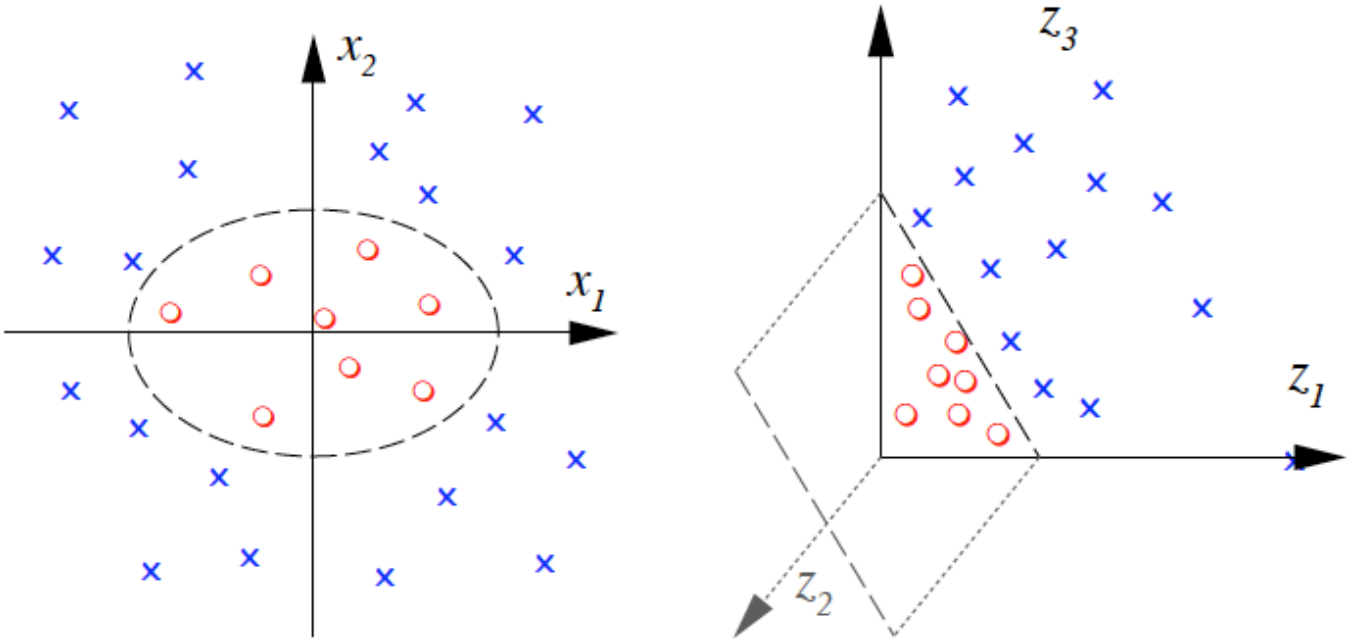
$$f(x) = w \cdot \Phi(x) + b.$$



Nonlinear Support Vector Machines – polynomial mapping

$$\Phi : R^2 \rightarrow R^3$$

$$(x_1, x_2) \mapsto (z_1, z_2, z_3) := (x_1^2, \sqrt{(2)} x_1 x_2, x_2^2)$$



Nonlinear Support Vector Machines – kernel tricks

Problem: the dimensionality of $\Phi(x)$ can be very large, making w hard to represent explicitly in memory, and hard for the QP to solve.

Once we obtain w and b, for an input x, the prediction can be:

$$w^T x + b = \left(\sum_{i=1}^{m} \alpha_i y^{(i)} x^{(i)} \right)^T x + b \tag{12}$$

$$= \sum_{i=1}^{m} \alpha_i y^{(i)} \langle x^{(i)}, x \rangle + b. \tag{13}$$

By examining the dual form of the optimization problem, we were also able to write the entire algorithm in terms of only inner products between input feature vectors. Thus we can replace all dot products with Kernel K(x, z) directly, and optimize alpha in dual, without the need to explicitly find $\phi(x)$

$$\max_\alpha \quad W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j \langle x^{(i)}, x^{(j)} \rangle$$

$$\text{s.t.} \quad 0 \le \alpha_i \le C, \quad i = 1, \ldots, m$$

$$\sum_{i=1}^{m} \alpha_i y^{(i)} = 0,$$

Nonlinear Support Vector Machines – Example

Let's see an example. Suppose $x, z \in \mathbb{R}^n$, and consider

$$K(x,z) = (x^T z)^2.$$

$$\begin{aligned} K(x,z) &= \left(\sum_{i=1}^{n} x_i z_i\right)\left(\sum_{j=1}^{n} x_i z_i\right) \\ &= \sum_{i=1}^{n}\sum_{j=1}^{n} x_i x_j z_i z_j \\ &= \sum_{i,j=1}^{n} (x_i x_j)(z_i z_j) \end{aligned}$$

Thus, we see that $K(x,z) = \phi(x)^T \phi(z)$, where

$$\phi(x) = \begin{bmatrix} x_1 x_1 \\ x_1 x_2 \\ x_1 x_3 \\ x_2 x_1 \\ x_2 x_2 \\ x_2 x_3 \\ x_3 x_1 \\ x_3 x_2 \\ x_3 x_3 \end{bmatrix}.$$

for the case of $n = 3$

calculating the high-dimensional $\phi(x)$ requires $O(n^2)$ time.

finding $K(x,z)$ takes only $O(n)$ time

Nonlinear Support Vector Machines – kernel tricks

$$\begin{aligned}
\max_\alpha \quad & W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j \langle x^{(i)}, x^{(j)} \rangle \\
\text{s.t.} \quad & 0 \le \alpha_i \le C, \quad i = 1, \ldots, m \\
& \sum_{i=1}^{m} \alpha_i y^{(i)} = 0,
\end{aligned}$$

$$\begin{aligned}
w^T x + b &= \left(\sum_{i=1}^{m} \alpha_i y^{(i)} x^{(i)} \right)^T x + b && (12) \\
&= \sum_{i=1}^{m} \alpha_i y^{(i)} \langle x^{(i)}, x \rangle + b. && (13)
\end{aligned}$$

Now, let's talk about a slightly different view of kernels. Intuitively, (and there are things wrong with this intuition, but nevermind), if $\phi(x)$ and $\phi(z)$ are close together, then we might expect $K(x, z) = \phi(x)^T \phi(z)$ to be large. Conversely, if $\phi(x)$ and $\phi(z)$ are far apart—say nearly orthogonal to each other—then $K(x, z) = \phi(x)^T \phi(z)$ will be small. So, we can think of $K(x, z)$ as some measurement of how similar are $\phi(x)$ and $\phi(z)$, or of how similar are x and z.

Nonlinear SVM – Polynomial kernel

The kernel $K(\boldsymbol{x}, \boldsymbol{x}') = (\boldsymbol{x} \cdot \boldsymbol{x}')^d$ gives the same result as the explicit mapping + dot product that we described before:

$$\Phi : R^2 \rightarrow R^3 \quad (x_1, x_2) \mapsto (z_1, z_2, z_3) := (x_1^2, \sqrt{(2)} x_1 x_2, x_2^2)$$

$$\Phi((x_1, x_2) \cdot \Phi((x'_1, x'_2) = (x_1^2, \sqrt{(2)} x_1 x_2, x_2^2) \cdot ({x'}_1^2, \sqrt{(2)} x'_1 x'_2, {x'}_2^2)$$

$$= x_1^2 {x'}_1^2 + 2 x_1 x'_1 x_2 x'_2 + x_2^2 {x'}_2^2$$

is the same as:

$$K(\boldsymbol{x}, \boldsymbol{x}') = (\boldsymbol{x} \cdot \boldsymbol{x}')^2 = ((x_1, x_2) \cdot (x'_1, x'_2))^2$$

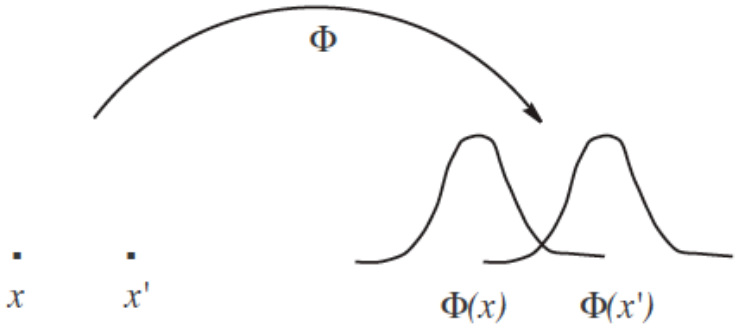
$$= (x_1 x'_1 + x_2 x'_2)^2 = x_1^2 {x'}_1^2 + x_2^2 {x'}_2^2 + 2 x_1 x'_1 x_2 x'_2$$

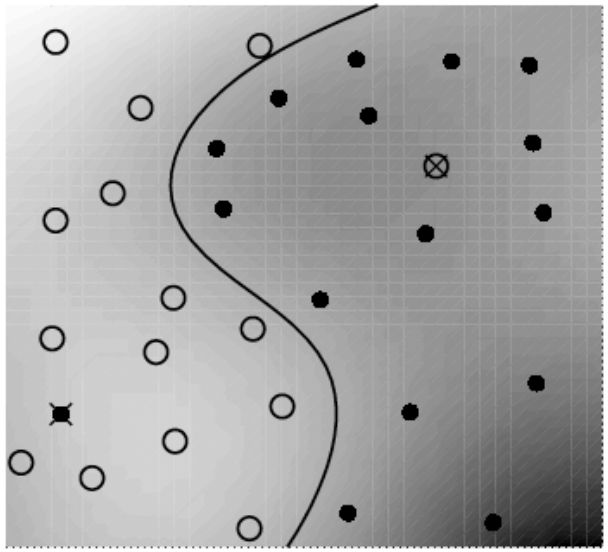
Interestingly, if d is large the kernel is still only requires n multiplications to compute, whereas the explicit representation may not fit in memory!

Nonlinear SVM – RBF kernel

The RBF kernel $K(\boldsymbol{x}, \boldsymbol{x}') = \exp(-\gamma||\boldsymbol{x} - \boldsymbol{x}'||^2)$ is one of the most popular kernel functions. It adds a "bump" around each data point:

$$f(\boldsymbol{x}) = \sum_{i=1}^{m} \alpha_i \exp(-\gamma||\boldsymbol{x}_i - \boldsymbol{x}||^2) + b$$





Using this one can get state-of-the-art results.

Linear Support Vector Machines – kernel tricks

Theorem (Mercer). Let $K : \mathbb{R}^n \times \mathbb{R}^n \mapsto \mathbb{R}$ be given. Then for K to be a valid (Mercer) kernel, it is necessary and sufficient that for any $\{x^{(1)}, \ldots, x^{(m)}\}$, $(m < \infty)$, the corresponding kernel matrix is symmetric positive semi-definite.

$$
\begin{aligned}
z^T K z &= \sum_i \sum_j z_i K_{ij} z_j \\
&= \sum_i \sum_j z_i \phi(x^{(i)})^T \phi(x^{(j)}) z_j \\
&= \sum_i \sum_j z_i \sum_k \phi_k(x^{(i)}) \phi_k(x^{(j)}) z_j \\
&= \sum_k \sum_i \sum_j z_i \phi_k(x^{(i)}) \phi_k(x^{(j)}) z_j \\
&= \sum_k \left(\sum_i z_i \phi_k(x^{(i)}) \right)^2 \\
&\geq 0.
\end{aligned}
$$

SVMs: software

Lots of SVM software:

- LibSVM (C++) https://www.csie.ntu.edu.tw/~cjlin/libsvm/
- SVMLight (C)

As well as complete machine learning toolboxes that include SVMs:

- Torch (C++)
- Spider (Matlab)
- Weka (Java)

Practice SVM on the basketball problem

How to use svm softwares

Chih-Wei Hsu, Chih-Chung Chang, and Chih-Jen Lin, A Practical Guide to Support Vector Classication. 2010.

Basketball Dataset

- We have,
  - ✓ a training/validation set: more than 80,000 negatives and 465 positives for 2-frame samples, and more than 80,000 negatives and 586 positives for 1-frame samples. 80 percentage is treated as the training set and 20 percentage is put to validation set.
  - ✓ a testing set: 158 positives and 28451 negatives for 2-frame samples and similar number of samples for the 1-frame case.

Libsvm

Download: https://www.csie.ntu.edu.tw/~cjlin/libsvm/

Guide:

Chih-Wei Hsu, Chih-Chung Chang, and Chih-Jen Lin, A Practical Guide to Support Vector Classication. 2010.

```
Usage: svm-train [options] training_set_file [model_file]
options:
-s svm_type : set type of SVM (default 0)
  0 -- C-SVC    (multi-class classification)
  1 -- nu-SVC   (multi-class classification)
  2 -- one-class SVM
  3 -- epsilon-SVR  (regression)
  4 -- nu-SVR   (regression)
-t kernel_type : set type of kernel function (default 2)
  0 -- linear: u'*v
  1 -- polynomial: (gamma*u'*v + coef0)^degree
  2 -- radial basis function: exp(-gamma*|u-v|^2)
  3 -- sigmoid: tanh(gamma*u'*v + coef0)
  4 -- precomputed kernel (kernel values in training_set_file)
-d degree : set degree in kernel function (default 3)
-g gamma : set gamma in kernel function (default 1/num_features)
-r coef0 : set coef0 in kernel function (default 0)
-c cost : set the parameter C of C-SVC, epsilon-SVR, and nu-SVR (default 1)
-n nu : set the parameter nu of nu-SVC, one-class SVM, and nu-SVR (default 0.5)
-p epsilon : set the epsilon in loss function of epsilon-SVR (default 0.1)
-m cachesize : set cache memory size in MB (default 100)
-e epsilon : set tolerance of termination criterion (default 0.001)
-h shrinking : whether to use the shrinking heuristics, 0 or 1 (default 1)
-b probability_estimates : whether to train a SVC or SVR model for probability estimates, 0 or 1 (default 0)
-wi weight : set the parameter C of class i to weight*C, for C-SVC (default 1)
-v n: n-fold cross validation mode
-q : quiet mode (no outputs)
```

Considerations in practice

- How to manage input feature vectors? Normalization or not?
- How to determine parameter C? crossvalidation
- How to choose kernel? Polynomial or RBF?
- How to determine the parameters in the kernel? Crossvalidation again

How to manage input feature vectors (linear case)? normalize or not?

Scaling before applying SVM is very important. Part 2 of Sarle's Neural Networks FAQ Sarle (1997) explains the importance of this and most of considerations also apply to SVM. The main advantage of scaling is to avoid attributes in greater numeric ranges dominating those in smaller numeric ranges. Another advantage is to avoid numerical difficulties during the calculation. Because kernel values usually depend on the inner products of feature vectors, e.g. the linear kernel and the polynomial kernel, large attribute values might cause numerical problems. We recommend linearly scaling each attribute to the range $[-1, +1]$ or $[0, 1]$.

How to determine parameter C (linear case)? crossvalidation

$$\min_{w,b} \quad \frac{1}{2}||w||^2 + C\sum_{i=1}^{m}\xi_i$$

Cross-validation and Grid-search

$$C = 2^{-5}, 2^{-3}, \ldots, 2^{15},$$

In v-fold cross-validation, we first divide the training set into v subsets of equal size. Sequentially one subset is tested using the classifier trained on the remaining $v-1$ subsets. Thus, each instance of the whole training set is predicted once so the cross-validation accuracy is the percentage of data which are correctly classified.

```
def DoCrossValiation(self):
    # self.PrintAll()
    y, x = self.load_fea_label()
    x = x[ : self.m_nMaximumSamples]
    y = y[ : self.m_nMaximumSamples]
    print('begin to convert y, x to label, dic ... ')
    b = time.clock()
    self.cvt_y_x_to_label_dic(y, x)
    e = time.clock()
    print('time elapsed in converting: %f' % (e - b))
    del y, x
    gc.collect()

    r = range(-5, 6, 1)
    fr = [2 ** x for x in r]
    print(fr)
    outinfo = []
    for each in fr:
        para = '-c ' + str(each) + ' -v 5'
        acc = train(self.m_labelList, self.m_dataDicList, para)
        print(para + ' acc = %f' % acc)
        outinfo += [para + ' acc = %f' % acc]
        print('time elapsed in training SVM: %f' % time.clock())
    print('result: ')
    [print(x) for x in outinfo]
```

How to determine parameter C (linear case)? crossvalidation

$$\min_{w,b} \quad \frac{1}{2}||w||^2 + C\sum_{i=1}^{m}\xi_i$$

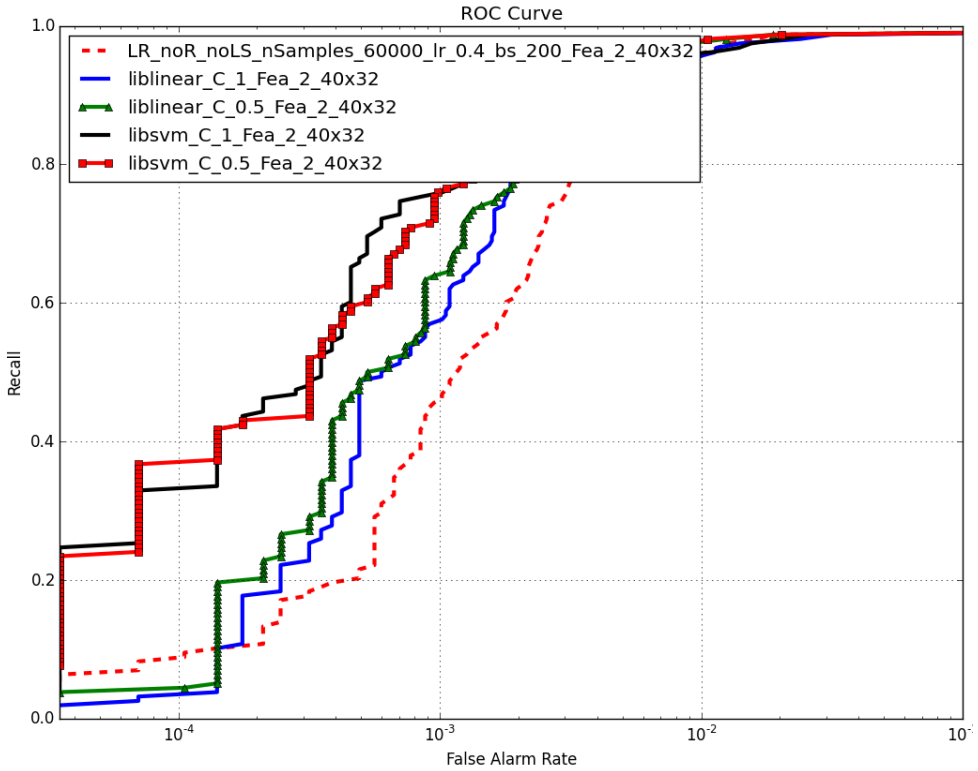
Cross-validation and Grid-search

$$C = 2^{-5}, 2^{-3}, \ldots, 2^{15},$$

In v-fold cross-validation, we first divide the training set into v subsets of equal size. Sequentially one subset is tested using the classifier trained on the remaining $v-1$ subsets. Thus, each instance of the whole training set is predicted once so the cross-validation accuracy is the percentage of data which are correctly classified.

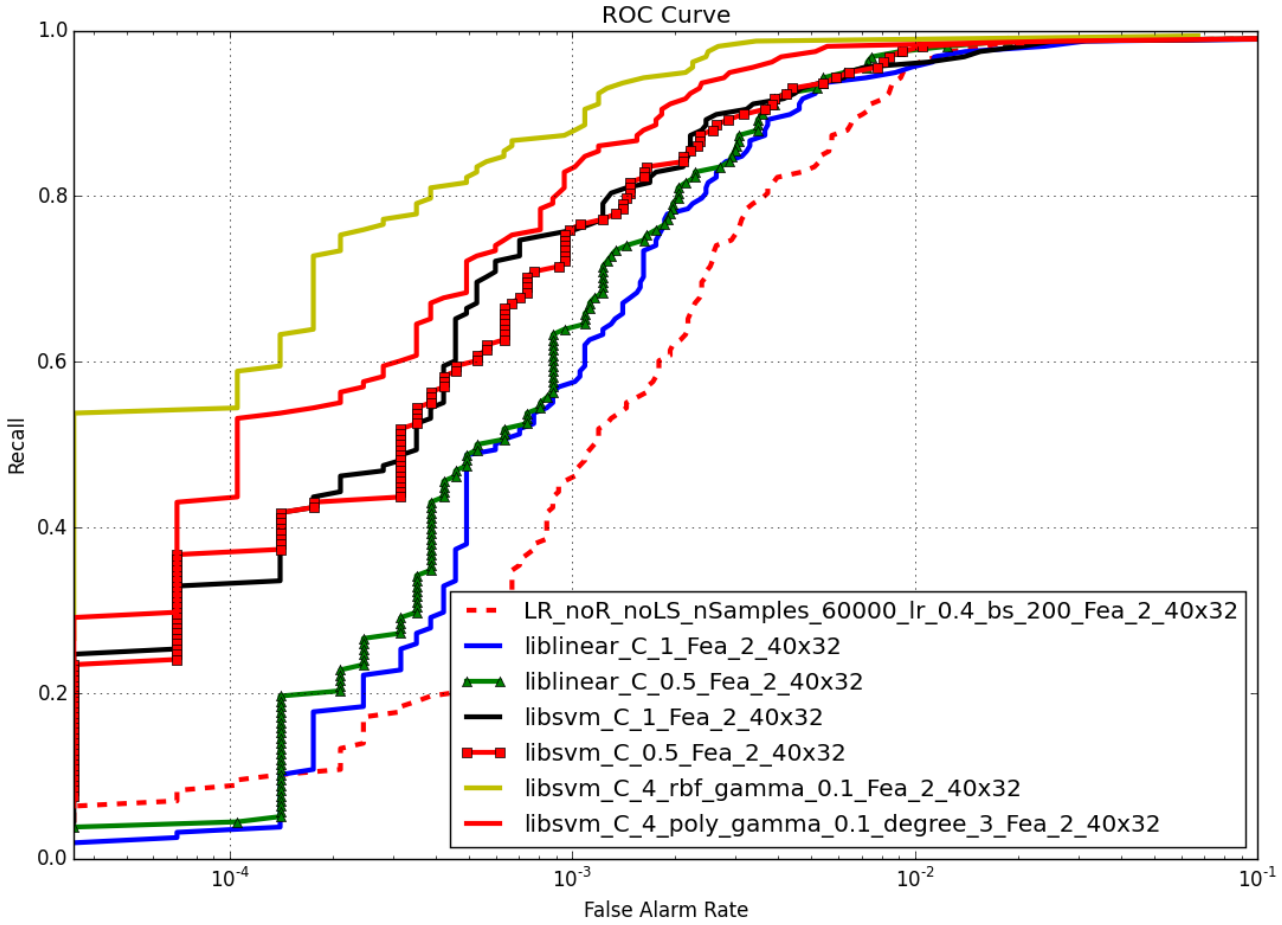
```
-c 0.03125 -v 5 acc = 99.778786
-c 0.0625 -v 5 acc = 99.829740
-c 0.125 -v 5 acc = 99.845896
-c 0.25 -v 5 acc = 99.855838
-c 0.5 -v 5 acc = 99.858323
-c 1 -v 5 acc = 99.840925
-c 2 -v 5 acc = 99.853352
-c 4 -v 5 acc = 99.837196
-c 8 -v 5 acc = 99.840925
-c 16 -v 5 acc = 99.833468
-c 32 -v 5 acc = 99.843410
```

The best C is 0.5



How to choose kernel? Polynomial or RBF?

- Polynomial kernel: $K(\boldsymbol{x}, \boldsymbol{x}') = (\boldsymbol{x} \cdot \boldsymbol{x}' + 1)^d$.
- RBF kernel: $K(\boldsymbol{x}, \boldsymbol{x}') = \exp(-\gamma||\boldsymbol{x} - \boldsymbol{x}'||^2)$.



The kernels are very powerful, especially RBF.

How to determine the parameters in the kernel? Crossvalidation again

How to choose the optimal C and gamma in RBF-kernel svm?

Cross-validation and Grid-search

We found that trying exponentially growing sequences of C and γ is a practical method to identify good parameters (for example, $C = 2^{-5}, 2^{-3}, \ldots, 2^{15}$, $\gamma = 2^{-15}, 2^{-13}, \ldots, 2^{3}$).

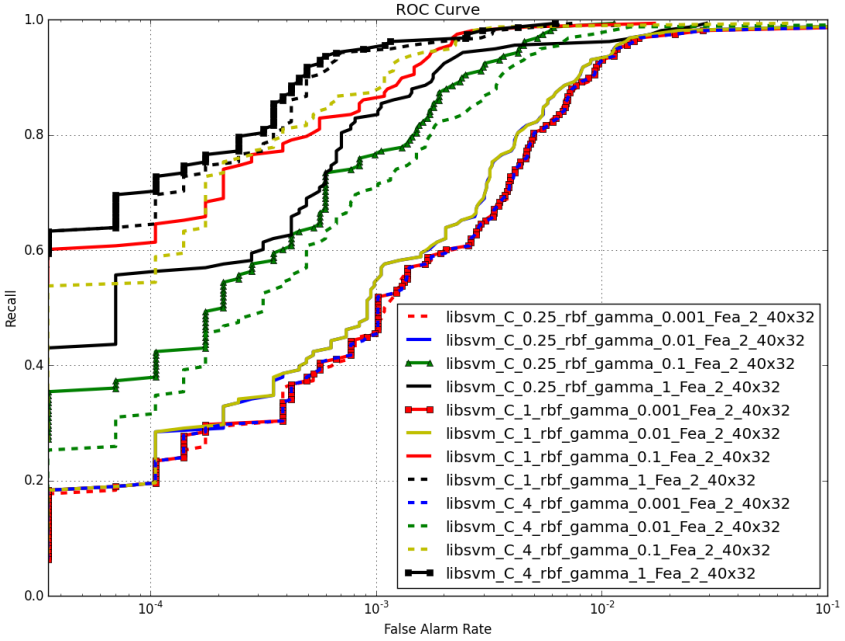
Since doing a complete grid-search may still be time-consuming, we recommend using a coarse grid first. After identifying a "better" region on the grid, a finer grid search on that region can be conducted.

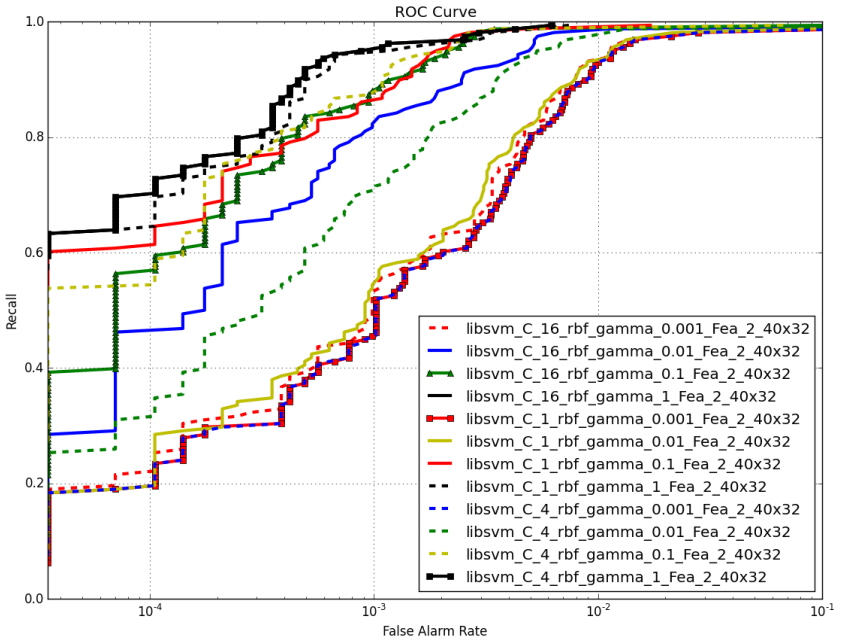
Because of the large number of samples in the problem, the search cost is highly expensive (e.g., 10 hours). here I only search C = 0.25, 1, 4, 16, and gamma = 0.001, 0.01, 0.1, 1, thus, 16 in total.

How to determine the parameters in the kernel? Crossvalidation again

How to choose the optimal C and gamma in RBF-kernel svm?

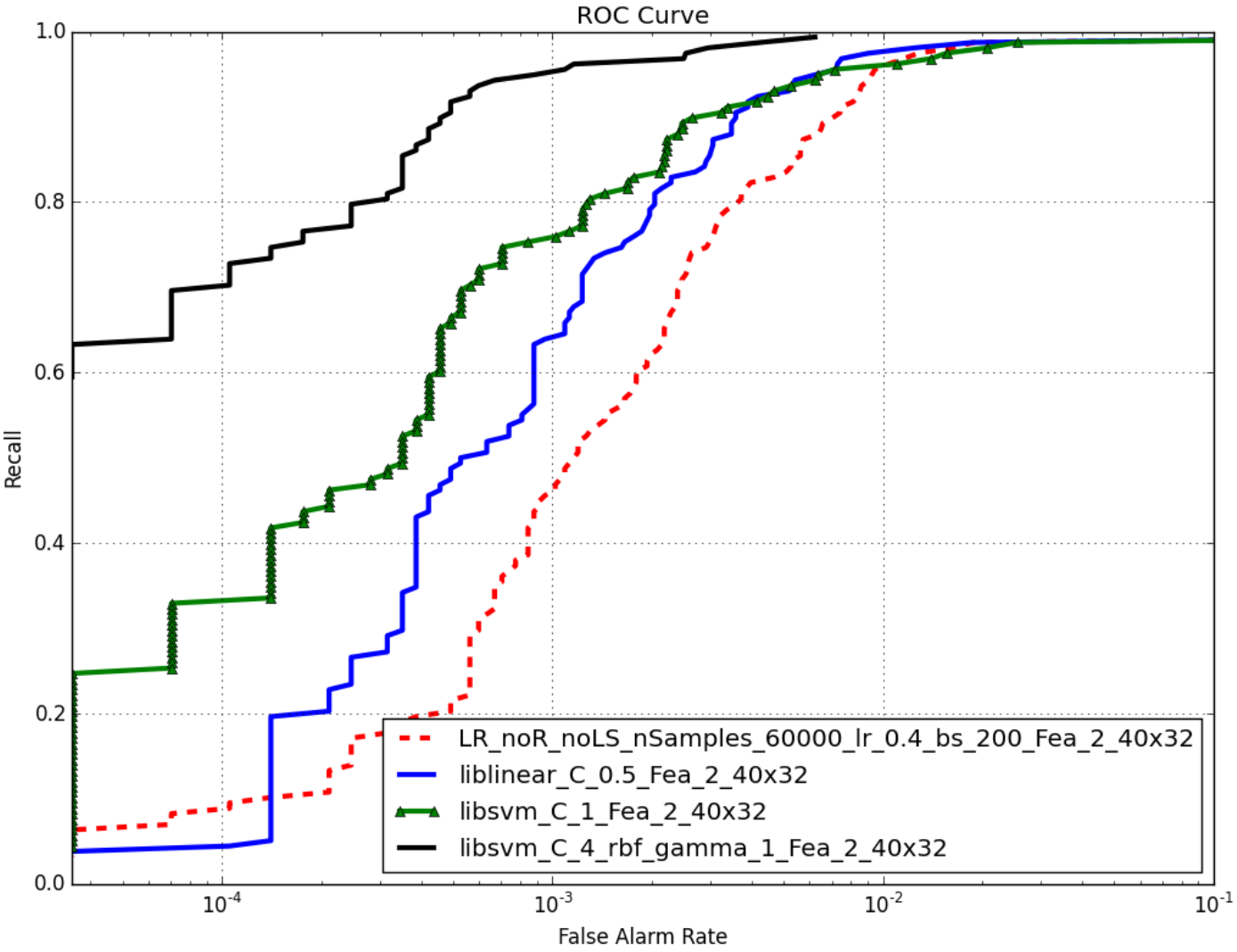
Because of the large number of samples in the problem, the search cost is still expensive (e.g., 10 hours). here I only search C = 0.25, 1, 4, 16, and gamma = 0.001, 0.01, 0.1, 1, in total.





Final results

The best LR, best linear svm (liblinear), best linear svm (libsvm), best RBF svm



Reference

CHRISTOPHER J.C. BURGES, A Tutorial on Support Vector Machines for Pattern Recognition, Data Mining and Knowledge Discovery 2, 121-167, 1998.

Andrew Ng, Support Vector Machines, CS229 Lecture notes3, Stanford University.

Chih-Wei Hsu, Chih-Chung Chang, and Chih-Jen Lin, A Practical Guide to Support Vector Classication. 2010.

Assignments

- Read the reference, especially Andrew Ng's note, and derive every formula by yourself in the slides

- Download libsvm and liblinear and learn to use them

- Carry out all experiments I showed on the basketball dataset.